Input Events Library Reference

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

Electronic edition published: Wednesday, October 8, 2014

Table of Contents

Touch Events

The Input Events library provides the framework necessary for Screen Graphics Subsystem to communicate with your touch driver.

What are touch events?

Touch events are events generated by the touch controller. Screen processes the events from the touch controller through callback functions provided by the touch driver.

Applications obtain access to information on the touch events with the use of the helper function *screen_get_mtouch_event()*.

These touch events are represented in the Input Events library by the data structure mtouch_event_t.

Information associated with a touch event that is included in `mtouch _event_t` are:

- Type of touch event (e.g., touch, release, move)
- Timestamp of touch event
- Sequence ID of touch event
- *x* and *y* coordinates of the touch event
- width and height of area of the touch event
- Orientation of the touch event
- Pressure of the touch event
- Contact type of the touch event

The *Input Events Library Reference* is intended for application developers. This table may help you find what you need in this reference:

For information about:	See:
Writing your own touch driver	*Writing a touch driver* (p. 11)
Writing callback functions	*Provide the callback functions* (p. 12)
Connecting to the Input Events Library	*Connect to the Input Events library* (p. 14)
Communicating with the touch-controller hardware	*Communicate with the hardware* (p. 17)
Disconnecting from the Input Events Library	*Disconnect from the Input Events library* (p. 18)
Input Events API	*Input Events Library Overview*

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	`NULL`
Data types	`unsigned short`
Environment variables	***PATH***
File and pathnames	`/dev/null`
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective** **Show View**.

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (*www.qnx.com*). You'll find a wide range of support options, including community forums.

Chapter 1
Touch Events

The Input Events library provides the framework necessary for Screen Graphics Subsystem to communicate with your touch driver.

What are touch events?

Touch events are events generated by the touch controller. Screen processes the events from the touch controller through callback functions provided by the touch driver.

Applications obtain access to information on the touch events with the use of the helper function *screen_get_mtouch_event()*.

These touch events are represented in the Input Events library by the data structure mtouch_event_t.

Information associated with a touch event that is included in `mtouch _event_t` are:

- Type of touch event (e.g., touch, release, move)
- Timestamp of touch event
- Sequence ID of touch event
- *x* and *y* coordinates of the touch event
- width and height of area of the touch event
- Orientation of the touch event
- Pressure of the touch event
- Contact type of the touch event

The *Input Events Library Reference* is intended for application developers. This table may help you find what you need in this reference:

For information about:	See:
Writing your own touch driver	*Writing a touch driver* (p. 11)
Writing callback functions	*Provide the callback functions* (p. 12)
Connecting to the Input Events Library	*Connect to the Input Events library* (p. 14)
Communicating with the touch-controller hardware	*Communicate with the hardware* (p. 17)
Disconnecting from the Input Events Library	*Disconnect from the Input Events library* (p. 18)
Input Events API	*Input Events Library Overview*

Chapter 2
Writing a Touch Driver

In order for Screen to be able to detect events from your touch controller, you need to provide a driver for your touch controller.

Your driver is required to provide the following:

Provide the required callback functions

Your driver must provide implementation for the callback functions: *get_coords()*, *get_contact_id()*, and *is_contact_down()*. Other callback functions that need to be implemented depend on the capabilities that you have specfied being supported by your driver.

Connect to the Input Events library

You need to specify the appropriate callback functions and the `mtouch` driver parameters for your touch controller. Once the callback functions and parameters have been set, you can connect to the Input Events library by calling the function *mtouch_driver_attach()*.

Communicate with the hardware

You need to communicate directly with the hardware via the hardware interface (e.g., I2C, SPI, USB, etc.).

Implement initialization and cleanup callback functions

Your driver must provide implementation for the callback functions: *mtouch_driver_init()* and *mtouch_driver_fini()*. These functions are your driver's insertion and exit points, respectively, for Screen.

Disconnect from the Input Events library

You need to perform the appropriate cleanup of resources and disconnect from the Input Events library by calling the function *mtouch_driver_detach()*.

Provide the callback functions

A set of functions need to be implemented by your touch driver. These callback functions are called into by the Input Events library to fetch the data required to transform the device (touch controller) data into mtouch events.

The driver will not be able to connect to the Input Events library unless the following functions are implemented:

get_contact_id()

Retrieves the contact ID for the specified digit of a touch-related event.

get_coords()

Retrieves the coordinates for the specified digit of a touch-related event.

is_contact_down()

Retrieves the touch status for the specified digit of a touch-related event.

You will also need to provide implementation for the callback functions that are related to the capabilities that are supported by your driver.

get_down_count()

Retrieves the number of touchpoints currently in contact with the screen. If you specify that your driver can track the number of touchpoints in contact with the screen (`MTOUCH_CAPABILITIES_CONTACT_COUNT`), then you need to implement this callback function in your touch driver.

get_touch_width()

Retrieves the width information of the touch area for the specified digit of a touch-related event. If you specify that your driver can provide the width of the touch area (`MTOUCH_CAPABILITIES_WIDTH`), then you need to implement this callback function in your touch driver.

get_touch_height()

Retrieves the height information of the touch area for the specified digit of a touch-related event. If you specify that your driver can provide the height of the touch area (`MTOUCH_CAPABILITIES_HEIGHT`), then you need to implement this callback function in your touch driver.

get_touch_pressure()

Retrieves the touch pressure information for the specified digit of a touch-related event. If you specify that your driver can provide the touch pressure (MTOUCH_CAPABILITIES_PRESSURE), then you need to implement this callback function in your touch driver.

get_seq_id()

Retrieves the unique sequence ID for the specified digit of a touch-related event. If you specify that your driver can provide sequence IDs (MTOUCH_CAPABILITIES_SEQ_ID) associated with a touch-related event, then you need to implement this callback function in your touch driver.

get_event_rate()

Retrieves the sampling period for the specified digit of a touch-related event. If you specify that your driver supports the capability of setting the event rate (MTOUCH_CAPABILITIES_RATE_SET), then you need to implement this callback function in your touch driver.

get_contact_type()

Retrieves the contact type of a touch-related event. If you specify that your driver can provide the type of contact (MTOUCH_CAPABILITIES_CONTACT_TYPE), then you need to implement this callback function in your touch driver.

Connect to the Input Events library

Callback functions and capabilities of your touch controller need to be configured and specified, respectively, before connecting to the Input Events library.

Configure callback functions

Assign your driver's callback functions appropriately by using the structure `mtouch_driver_funcs_t`.

For example:

```
mtouch_driver_funcs_t funcs = {
 .get_contact_id = get_contact_id,
 .is_contact_down = is_contact_down,
 .get_coords = get_coords,
 .get_down_count = get_down_count,
 .get_touch_width = NULL,
 .get_touch_height = NULL,
 .get_touch_orientation = NULL,
 .get_touch_pressure = NULL,
 .get_seq_id = NULL,
 .set_event_rate = NULL,
 .get_contact_type = NULL,
 .get_select = NULL
};
```

Specify driver parameters

Specify the capabilities that are supported by your driver. Use the structure `mtouch_driver_params_t`.

For example:

```
mtouch_driver_params_t params = {
 .capabilities = MTOUCH_CAPABILITIES_CONTACT_ID |
                 MTOUCH_CAPABILITIES_COORDS |
                 MTOUCH_CAPABILITIES_CONTACT_COUNT,
 .flags = 0,
 .max_touchpoints = 2,
 .width = 1312,
 .height = 800
};
```

Connect to the Input Events library

After having assigned your callback functions and having specified your driver parameters, you use the function *mtouch_driver_attach()* to connect to the Input Events library.

For example:

```
struct mtouch_device* inputevents_hdl;
inputevents_hdl = mtouch_driver_attach(&params, &funcs);
```

Provide initialization and cleanup callback functions

Screen makes two function calls in the driver: one for initialization and one for cleanup.

Screen parses the configuration file, `graphics.conf`, to determine the driver specified in the `mtouch` section of this configuration file.

Screen calls *dlopen()* on this specified driver. Upon a successful *dlopen()*, Screen will use *dlsym()* to look for two entries: *mtouch_driver_init()* and *mtouch_driver_fini()*.

Implement initialization callback function

mtouch_driver_init() performs the following:

1. Sets the default values of the driver.
2. Parses any options specified in `graphics.conf` using *input_parseopts()*. See Configure `mtouch`.
3. Performs any necessary hardware initialization.
4. Configure callback functions using `mtouch_driver_funcs_t` as described in *Connect to the Input Events library* (p. 14).
5. Specify driver parameters using `mtouch_driver_params_t` as described in *Connect to the Input Events library* (p. 14).
6. Connect to Input Events library as described in *Connect to the Input Events library* (p. 14).
7. Create a separate thread to communicate directly with the hardware via the hardware interface (e.g., I2C, SPI, USB, etc.) and trigger the Input Events library API function *mtouch_driver_process_packet()* to start processing the touch-related event data.

The driver insertion point for Screen is *mtouch_driver_init()*. This callback function has the following signature:

```
void* (*init)(const char* options);
```

An example implementation of *mtouch_driver_init()*:

```
void * mtouch_driver_init(const char* options)
{
 pthread_attr_t pattr;
 sched_param_t param;

 /* touch_dev_t is the structure which contains information specific to
  * your touch device along with other NTO and thread information.
  */
 touch_dev_t *td = (touch_dev_t *)calloc(1,sizeof(touch_dev_t));
    memset(td,0,sizeof(touch_dev_t));
 if(td==NULL){
  mtouch_error("touch_device", "Failed to allocate memory for device structure");
  return NULL;
 }

 /* Initialize any defaults for your touch device here */
 td->thread_priority = 21; /* Thread priority for communicating directly with the hardware */

   /* Parse the mtouch options from graphics.conf */
 input_parseopts(options, set_options, td);

   /* Connect to device */
 td->chid = ChannelCreate(0);
```

```
 if(td->chid == -1) {
  mtouch_error("touch_device","%s: ChannelCreate: %s",__FUNCTION__,strerror(errno));
  goto failChannelCreate;
 }

 td->coid = ConnectAttach(0,0,td->chid,_NTO_SIDE_CHANNEL,0);
 if(td->coid==-1) {
  mtouch_error("touch_device","%s: ConnectAttach: %s",__FUNCTION__,strerror(errno));
  goto failConnectAttach;
 }

 /* Utility function attach_drier() to configure your callback function and
  *  connect to the Input Events library
  */
 if(attach_driver(td)!=EOK) {
  goto failAttachLibInputEvents;
 }

 pthread_attr_init(&pattr);
 param.sched_priority = td->thread_priority;
 pthread_attr_setschedparam(&pattr, &param);
 pthread_attr_setinheritsched(&pattr, PTHREAD_EXPLICIT_SCHED);

 /* Create a thread for communicating directly with the hardware and to
  * trigger mtouch_driver_process_packet() */
 int ret = pthread_create(&td->recv_thread, &pattr, tp_recv_thread, td);
 if (EOK != ret) {
  mtouch_error("touch_device", "Failed to create the intr thread (%s - %i)",strerror(errno),ret);
  goto failCreateThread;
 }
 pthread_setname_np(td->recv_thread, "touch_device");

 return td;

failCreateThread:
 InterruptDetach(td->tp_iid);
 td->tp_iid = -1;

failAttachLibInputEvents:
 mtouch_driver_detach(td->inputevents_hdl);
 td->inputevents_hdl = NULL;

failInitialize:
 ChannelDestroy(td->chid);
 td->chid = -1;

failConnectAttach:
 ConnectDetach(td->chid);
 td->chid = -1;

failChannelCreate:
 free(td);
 return NULL;
}
```

Implement cleanup callback function

mtouch_driver_fini() performs any necessary device cleanup and disconnects from the Input Events library.

The driver exit point for Screen is *mtouch_driver_fini()*. This callback function has the following signature:

```
void (*fini)(void* dev);
```

An example implementation of *mtouch_driver_fini()*:

```
void mtouch_driver_fini(void* dev)
{
 touch_dev_t *td = dev;

 pthread_cancel(td->tp_recv_thread);
 pthread_join(td->tp_recv_thread, NULL);

 if (td->inputevents_hdl) {
  mtouch_driver_detach(td->inputevents_hdl);
  td->inputevents_hdl = NULL;
 }

 free(td);
}
```

Communicate with the hardware

A separate thread is used to communicate directly with the hardware via the hardware interface (e.g., I2C, SPI, USB, etc.).

You need to create a separate thread as part of your implementation of the *mtouch_driver_init()* callback function. Refer to *pthread_create()* in the *QNX Neutrino C Library Reference* for more details.

The routine you provide as the argument to *pthread_create()* needs to retrieve coordinate information from the hardware and then call *mtouch_driver_process_packet()* to trigger the start or processing the data for the touch-related event.

Disconnect from the Input Events library

In addition to any device cleanup, you need to disconnect from the Input Events library so that any resources allocated by the framework to support your driver are released.

At any exit point of your driver, you need to call *mtouch_driver_detach()* to release resources.

For example, you will need to call *mtouch_driver_detach()* from your implmentation of *mtouch_driver_fini()*.

```
void mtouch_driver_fini(void* dev)
{
 touch_dev_t *td = dev;

 pthread_cancel(td->tp_recv_thread);
 pthread_join(td->tp_recv_thread, NULL);

 if (td->inputevents_hdl) {
  mtouch_driver_detach(td->inputevents_hdl);
  td->inputevents_hdl = NULL;
 }

 free(td);
}
```

Chapter 3
Input Events Library Overview

The Input Events library allows applications to receive and process events from input devices. The classes of events that are recognized by the Input Events library are:

- TOUCH (INPUT_CLASS_MTOUCH)
- KEYBOARD (INPUT_CLASS_KEYBOARD)
- MOUSE (INPUT_CLASS_MOUSE)

event_types.h

Enumerations and structures for input events.

The `event_types.h` header file provides type definitions for classifying input event types. These type definitions can be used to determine the kind of input event that has occurred and the properties of the event.

Data types in *event_types.h*

Data structures, typedefs, and enumerations that are available for input events.

contact_type_e

Types of contact for an `INPUT_CLASS_MTOUCH` event.

Synopsis:

```
#include <input/event_types.h>


typedef enum {
      CONTACT_TYPE_FINGER = 0
      CONTACT_TYPE_STYLUS = 1
} contact_type_e;
```

Data:

CONTACT_TYPE_FINGER

Finger touch (default).

CONTACT_TYPE_STYLUS

Stylus touch.

Library:

`libinputevents`

Description:

input_class_e

Classes of input events.

Synopsis:

```
#include <input/event_types.h>


typedef enum {
      INPUT_CLASS_MTOUCH = 1
```

```
      INPUT_CLASS_KEYBOARD
      INPUT_CLASS_MOUSE
} input_class_e;
```

Data:

INPUT_CLASS_MTOUCH

A touch-related event on the screen.

INPUT_CLASS_KEYBOARD

A key event on the virtual keyboard.

INPUT_CLASS_MOUSE

A mouse event using a connected mouse.

Library:

`libinputevents`

Description:

input_event_e

Types of `INPUT_CLASS_MTOUCH` events.

Synopsis:

```
#include <input/event_types.h>


typedef enum {
      INPUT_EVENT_UNKNOWN = 0
      INPUT_EVENT_MTOUCH_TOUCH = 100
      INPUT_EVENT_MTOUCH_MOVE
      INPUT_EVENT_MTOUCH_RELEASE
      INPUT_EVENT_MTOUCH_CANCEL
} input_event_e;
```

Data:

INPUT_EVENT_UNKNOWN

Unknown event type (default).

INPUT_EVENT_MTOUCH_TOUCH

Event type for when there is new contact with the screen detected.

INPUT_EVENT_MTOUCH_MOVE

Event type for when contact with the screen is already detected, but the contact is changing position.

INPUT_EVENT_MTOUCH_RELEASE

Event type for when contact with the screen is removed.

INPUT_EVENT_MTOUCH_CANCEL

Event type for when a gesture is cancelled; a touch controller that is powered off could cause a cancel event.

Library:

`libinputevents`

Description:

One event type is received per touchpoint. For example, if you have two fingers in contact with the screen, you will receive two `INPUT_EVENT_MTOUCH_TOUCH` events. Similarly, if you remove two fingers from contact with the screen, you will receive two `INPUT_EVENT_MTOUCH_RELEASE` events, one for each finger. The touch-related events are identified individually through a contact ID.

mtouch_event

Structure that contains details common to `INPUT_CLASS_MTOUCH` input events.

Synopsis:

```
typedef struct mtouch_event {
    input_event_e event_type ;
    _Uint64t timestamp ;
    _Uint32t seq_id ;
    _Uint32t contact_id ;
    _Int32t x ;
    _Int32t y ;
    _Uint32t width ;
    _Uint32t height ;
    _Uint32t orientation ;
    _Uint32t pressure ;
    _Uint32t contact_type ;
    _Uint32t select ;
}mtouch_event_t;
```

Data:

input_event_e event_type

The event type of the INPUT_CLASS_MTOUCH event.

_Uint64t timestamp

Timestamp, based on realtime clock, when the INPUT_CLASS_MTOUCH event occurred.

_Uint32t seq_id

The sequence number for the event; seq_id is incremented each time a new touch-related event occurs.

_Uint32t contact_id

The order of occurrence for multiple touch contacts.

_Int32t x

The x screen position, in pixels, for the event.

_Int32t y

The y screen position, in pixels, for the event.

_Uint32t width

The width, in pixels, of the touch area.

_Uint32t height

The height, in pixels, of the touch area.

_Uint32t orientation

The orientation of the contact.

_Uint32t pressure

The pressure of the touch contact, ranging from 0 to 2^32 - 1.

_Uint32t contact_type

The contact type.

Valid contact types are of type *contact_type_e* (p. 20).

_Uint32t select

The selected buttons.

Library:

`libinputevents`

Description:

The *mtouch_event* (p. 22) structure represents information that is common to all input events of class `INPUT_CLASS_MTOUCH`. This information is provided by the input device driver whenever an `INPUT_CLASS_MTOUCH` event occurs.

mtouch_event_t

Touch event information.

Synopsis:

```
#include <input/event_types.h>

typedef struct mtouch_client_params  mtouch_client_params_t;
```

Library:

`libinputevents`

Description:

This type is an alias for an mtouch event, mtouch_event. Use this `typedef` when working with an mtouch event and its associated information.

mtouch_driver.h

Functions that are to be implemented by the mtouch driver and are called by libinputevents to fetch the data required to transform the device data into mtouch events.

Constants in *mtouch_driver.h*

Constants that are available for input events.

Definitions in *mtouch_driver.h*

Preprocessor macro definitions for the mtouch_driver.h header file in the libinputevents library.

Definitions:

`#define MTOUCH_CAPABILITIES_CONTACT_ID (1 << 0)`

The multitouch device capability to support contact IDs.

`#define MTOUCH_CAPABILITIES_COORDS (1 << 1)`

The multitouch device capability to support coordinates.

`#define MTOUCH_CAPABILITIES_CONTACT_COUNT (1 << 2)`

The multitouch device capability to support contact counts.

`#define MTOUCH_CAPABILITIES_WIDTH (1 << 3)`

The multitouch device capability to support touch widths.

`#define MTOUCH_CAPABILITIES_HEIGHT (1 << 4)`

The multitouch device capability to support touch heights.

`#define MTOUCH_CAPABILITIES_ORIENTATION (1 << 5)`

The multitouch device capability to support touch orientations.

`#define MTOUCH_CAPABILITIES_PRESSURE (1 << 6)`

The multitouch device capability to support touch pressures (or signal strength for capacitive touchscreens).

`#define MTOUCH_CAPABILITIES_RATE_SET (1 << 7)`

The multitouch device capability to support setting of an event rate.

`#define MTOUCH_CAPABILITIES_SEQ_ID (1 << 8)`

The multitouch device capability to support sequence IDs.

`#define MTOUCH_CAPABILITIES_CONTACT_TYPE (1 << 9)`

The multitouch device capability to support contact type.

#define MTOUCH_DEFAULT_WIDTH 1

The width when MTOUCH_CAPABILITIES_WIDTH isn't set or if this capability isn't available.

#define MTOUCH_DEFAULT_HEIGHT 1

The height when MTOUCH_CAPABILITIES_HEIGHT isn't set or if this capability isn't available.

#define MTOUCH_DEFAULT_PRESSURE 1

The pressure when MTOUCH_CAPABILITIES_PRESSURE isn't set or if this this capability isn't available.

#define MTOUCH_FLAGS_INCONSISTENT_DIGIT_ORDER (1 << 0)

A multi-touch device flag.

Devices that report touch data for individual fingers in an inconsistent order should set this flag. Use this flag, along with the MTOUCH_FLAGS_INCONSISTENT_CONTACT_IDS flag, to describe the relationship between the digit index and the contact ID.

#define MTOUCH_FLAGS_INCONSISTENT_CONTACT_IDS (1 << 1)

A multitouch device flag.

Devices that don't have contact IDs assigned as a zero-based index should set this flag. Use this flag along with the MTOUCH_FLAGS_INCONSISTENT_DIGIT_ORDER flag to describe the relationship between the digit index and the contact ID.

Library:

libinputevents

Data types in *mtouch_driver.h*

Data structures, typedefs, and enumerations that are available for input events.

mtouch_driver_funcs_t

Functions to be implemented by the mtouch driver.

Synopsis:

```
typedef struct  {
    int(* get_contact_id )(void *packet, _Uint8t digit_idx, _Uint32t
*contact_id, void *arg);
    int(* is_contact_down )(void *packet, _Uint8t digit_idx, int *valid, void
 *arg);
    int(* get_coords )(void *packet, _Uint8t digit_idx, _Int32t *x, _Int32t
*y, void *arg);
    int(* get_down_count )(void *packet, _Uint32t *down_count, void *arg);
    int(* get_touch_width )(void *packet, _Uint8t digit_idx, _Uint32t
*touch_width, void *arg);
    int(* get_touch_height )(void *packet, _Uint8t digit_idx, _Uint32t
*touch_height, void *arg);
    int(* get_touch_orientation )(void *packet, _Uint8t digit_idx, _Uint32t
*touch_orientation, void *arg);
    int(* get_touch_pressure )(void *packet, _Uint8t digit_idx, _Uint32t
```

```
*touch_pressure, void *arg);
    void(* get_seq_id )(void *packet, _Uint32t *seq_id, void *arg);
    int(* get_contact_type )(void *packet, _Uint8t digit_idx, _Uint32t
*contact_type, void *arg);
   int(* get_select )(void *packet, _Uint8t digit_idx, _Uint32t *select, void
 *arg);
    int(* set_event_rate )(void *dev, _Uint32t min_event_interval);
}mtouch_driver_funcs_t;
```

Data:

int(* get_contact_id)(void *packet, _Uint8t digit_idx, _Uint32t *contact_id, void *arg)

Retrieve the contact ID for the specified digit from packet.

This callback function is called for each of the touchpoints. The maximum number of touchpoints is `max_touchpoints`, as specified in `mtouch_driver_params_t` (`mtouch_params.h`).

Arguments

- `packet` Data packet that contains information on the touch-related event. This data packet is the same packet that was passed into *mtouch_driver_process_packet() (p. 36)*. This data can be used to retrieve information about all digits, regardless of whether or not they are touching.
- `digit_idx` Digit (finger) index that the Input Events library is requesting. It is a zero based index whose maximum is (`max_touchpoints` - 1). There is no requirement for any correlation between `digit_idx` and `contact_id`.
- `contact_id` Pointer to contact ID of the touch-related event for the specified `digit_idx`. This function updates `contact_id` upon its successful execution.
- `arg` User information.

Returns

- `0` if successful; otherwise non-zero if an error occurred.

int(* is_contact_down)(void *packet, _Uint8t digit_idx, int *valid, void *arg)

Retrieve the touch status for the specified digit from packet.

This callback function is called for each of the touchpoints. The maximum number of touchpoints is `max_touchpoints`, as specified in `mtouch_driver_params_t` (`mtouch_params.h`).

Arguments

- `packet` Data packet that contains information on the touch-related event. This data packet is the same packet that was passed into

mtouch_driver_process_packet() (p. 36). This data can be used to retrieve information about all digits, regardless of whether or not they are touching.

- `digit_idx` Digit (finger) index that the Input Events library is requesting. It is a zero based index whose maximum is (`max_touchpoints` - 1). There is no requirement for any correlation between `digit_idx` and `contact_id`.
- `valid` Pointer to touch status (e.g., `1` = Down and `0` = Up) of the touch-related event for the specified `digit_idx`. This function updates `valid` upon its successful execution.
- `arg` User information.

Returns

- `0` if successful; otherwise non-zero if an error occurred.

int(* get_coords)(void *packet, _Uint8t digit_idx, _Int32t *x, _Int32t *y, void *arg)

Retrieve the coordinates for the specified digit from packet.

This callback function is called for each of the touchpoints. The maximum number of touchpoints is `max_touchpoints`, as specified in `mtouch_driver_params_t` (`mtouch_params.h`).

Arguments

- `packet` Data packet that contains information on the touch-related event. This data packet is the same packet that was passed into *mtouch_driver_process_packet() (p. 36)*. This data can be used to retrieve information about all digits, regardless of whether or not they are touching.
- `digit_idx` Digit (finger) index that the Input Events library is requesting. It is a zero based index whose maximum is (`max_touchpoints` - 1). There is no requirement for any correlation between `digit_idx` and `contact_id`.
- `x` Pointer to the x coordinate of the touch-related event for the specified `digit_idx`. This function updates `x` upon its successful execution.
- `y` Pointer to the y coordinate of the touch-related event for the specified `digit_idx`. This function updates `y` upon its successful execution.
- `arg` User information.

Returns

- `0` if successful; otherwise non-zero if an error occurred.

int(* get_down_count)(void *packet, _Uint32t *down_count, void *arg)

Retrieve the number of touchpoints currently in contact with the screen.

This callback function is called for each of the touchpoints. The maximum number of touchpoints is `max_touchpoints`, as specified in `mtouch_driver_params_t` (`mtouch_params.h`).

Arguments

- `packet` Data packet that contains information on the touch-related event. This data packet is the same packet that was passed into *mtouch_driver_process_packet() (p. 36)*. This data can be used to retrieve information about all digits, regardless of whether or not they are touching.
- `down_count` Pointer to the number of rouchpoints currently in contact with the screen. This function updates `down_count` upon its successful execution.
- `arg` User information.

Returns

- `0` if successful; otherwise non-zero if an error occurred.

int(* get_touch_width)(void *packet, _Uint8t digit_idx, _Uint32t *touch_width, void *arg)

Retrieve the width information for the specified digit from packet.

This callback function is called for each of the touchpoints. The maximum number of touchpoints is `max_touchpoints`, as specified in `mtouch_driver_params_t` (`mtouch_params.h`).

Arguments

- `packet` Data packet that contains information on the touch-related event. This data packet is the same packet that was passed into *mtouch_driver_process_packet() (p. 36)*. This data can be used to retrieve information about all digits, regardless of whether or not they are touching.
- `digit_idx` Digit (finger) index that the Input Events library is requesting. It is a zero based index whose maximum is (`max_touchpoints` - 1). There is no requirement for any correlation between `digit_idx` and `contact_id`.
- `touch_width` Pointer to the width of the touch-related event for the specified `digit_idx`. This function updates `touch_width` upon its successful execution.
- `arg` User information.

Returns

- `0` if successful; otherwise non-zero if an error occurred.

int(* get_touch_height)(void *packet, _Uint8t digit_idx, _Uint32t *touch_height, void *arg)

Retrieve the height information for the specified digit from packet.

This callback function is called for each of the touchpoints. The maximum number of touchpoints is `max_touchpoints`, as specified in `mtouch_driver_params_t` (`mtouch_params.h`).

Arguments

- `packet` Data packet that contains information on the touch-related event. This data packet is the same packet that was passed into *mtouch_driver_process_packet() (p. 36)*. This data can be used to retrieve information about all digits, regardless of whether or not they are touching.
- `digit_idx` Digit (finger) index that the Input Events library is requesting. It is a zero based index whose maximum is (`max_touchpoints` - 1). There is no requirement for any correlation between `digit_idx` and `contact_id`.
- `touch_height` Pointer to the height of the touch-related event for the specified `digit_idx`. This function updates `touch_height` upon its successful execution.
- `arg` User information.

Returns

- `0` if successful; otherwise non-zero if an error occurred.

int(* get_touch_orientation)(void *packet, _Uint8t digit_idx, _Uint32t *touch_orientation, void *arg)

Retrieve the orientation information for the specified digit from packet.

This callback function is not yet implemented by the Input Events library. For the time being, simply set this callback function to `NULL` in your *mtouch_driver_funcs_t* (p. 26) assignment before you call *mtouch_driver_attach() (p. 35)*.

Arguments

- `packet`
- `digit_idx`
- `touch_orientation`
- `arg`

Returns

- `0` if successful; otherwise non-zero if an error occurred.

int(* get_touch_pressure)(void *packet, _Uint8t digit_idx, _Uint32t *touch_pressure, void *arg)

Retrieve the touch pressure information for the specified digit from packet.

For capacitive touchscreens, this callback function retrieves the signal strength, not the pressure. The signal strength information is directly proportional to the width and height retrieved from the *(*get_touch_width)()* and *(*get_touch_height)()* callback functions.

Arguments

- `packet` Data packet that contains information on the touch-related event. This data packet is the same packet that was passed into *mtouch_driver_process_packet() (p. 36)*. This data can be used to retrieve information about all digits, regardless of whether or not they are touching.
- `digit_idx` Digit (finger) index that the Input Events library is requesting. It is a zero based index whose maximum is (`max_touchpoints` - 1). There is no requirement for any correlation between `digit_idx` and `contact_id`.
- `touch_pressure` Pointer to the pressure, or signal strength for capacitive touchscreens, of the touch-related event for the specified `digit_idx`. This function updates is `touch_pressure` upon its successful execution.
- `arg` User information.

Returns

- `0` if successful; otherwise non-zero if an error occurred.

void(* get_seq_id)(void *packet, _Uint32t *seq_id, void *arg)

Retrieve the unique sequence ID of a touch-related event from packet.

This callback function is called for each of the touchpoints. The maximum number of touchpoints is `max_touchpoints`, as specified in `mtouch_driver_params_t` (`mtouch_params.h`). The sequence ID is used to track the touch-related event in the Input Events library. The sequence ID is commonly a value that is incremented continuously.

Arguments

- `packet` Data packet that contains information on the touch-related event. This data packet is the same packet that was passed into

mtouch_driver_process_packet() (p. 36). This data can be used to retrieve information about all digits, regardless of whether or not they are touching.

- `seq_id` Pointer to the unique sequence ID that Input Events library is requesting. This function updates `seq_id` upon its successful execution.
- `arg` User information.

Returns

- `0` if successful; otherwise non-zero if an error occurred.

int(* get_contact_type)(void *packet, _Uint8t digit_idx, _Uint32t *contact_type, void *arg)

Retrieve the contact type of a touch-related event from packet.

This callback function is called for each of the touchpoints. The maximum number of touchpoints is `max_touchpoints`, as specified in `mtouch_driver_params_t` (`mtouch_params.h`).

Arguments

- `packet` Data packet that contains information on the touch-related event. This data packet is the same packet that was passed into *mtouch_driver_process_packet() (p. 36)*. This data can be used to retrieve information about all digits, regardless of whether or not they are touching.
- `digit_idx` Digit (finger) index that is related to the contact type that is being retrived. It is a zero based index whose maximum is(`max_touchpoints` - 1).
- `contact_type` Pointer to the contact type that Input Events library is requesting. Valid contact types are of type *contact_type_e* (p. 20). This function updates `contact_type` upon its successful execution.
- `arg` User information.

Returns

- `0` if successful; otherwise non-zero if an error occurred.

int(* get_select)(void *packet, _Uint8t digit_idx, _Uint32t *select, void *arg)

Retrieve the select buttons of a touch-related event from packet.

This callback function is not yet implemented by the Input Events library. For the time being, simply set this callback function to `NULL` in your *mtouch_driver_funcs_t* (p. 26) assignment when attaching your driver.

Arguments

- `packet`

- `digit_idx`
- `select`
- `arg`

Returns

- `0` if successful; otherwise non-zero if an error occurred.

int(* set_event_rate)(void *dev, _Uint32t min_event_interval)

Retrieve the sampling period of a touch-related event from packet.

This callback function is called for each of the touchpoints. The maximum number of touchpoints is `max_touchpoints`, as specified in `mtouch_driver_params_t` (`mtouch_params.h`). This function is called with the device mutex held.

Arguments

- `dev` Handle to the device driver.
- `min_event_interval` Minimum sampling period, in microseconds, of the touch controller. Two touch-related events will not be received in a time less than this interval.

Returns

- `0` if successful; otherwise non-zero if an error occurred.

Library:

`libinputevents`

Description:

These callback functions are called into by `libinputevents` to fetch the data required to transform the device (touch controller) data into mtouch events.

These functions must be implemented. Otherwise, the driver will not be able to connect to the Input Events framework.

Driver test result types

The types of test results.

Synopsis:

```
#include <input/mtouch_driver.h>

 enum {
      MTOUCH_TEST_RESULT_COMPLETED
```

```
        MTOUCH_TEST_RESULT_NOT_SUPPORTED
        MTOUCH_TEST_RESULT_NOT_NOW
        MTOUCH_TEST_RESULT_I2C_FAILURE
        MTOUCH_TEST_RESULT_TIMEOUT
};
```

Data:

MTOUCH_TEST_RESULT_COMPLETED

Test has completed.

MTOUCH_TEST_RESULT_NOT_SUPPORTED

Test is not supported.

MTOUCH_TEST_RESULT_NOT_NOW

Test cannot be run at this time.

Possible reasons include:

- firmware is being updated
- single scan
- BIST is occurring

MTOUCH_TEST_RESULT_I2C_FAILURE

I2C communication with controller failed.

MTOUCH_TEST_RESULT_TIMEOUT

The controller did not respond in time.

Library:

`libinputevents`

Description:

This enumeration lists possible test results from using the test interface for an mtouch driver.

Functions in *mtouch_driver.h*

Functions that are available for input events.

mtouch_driver_attach()

Attach driver to the Input Events framework.

Synopsis:

```
#include <input/mtouch_driver.h>

struct mtouch_device* mtouch_driver_attach(mtouch_driver_params_t *params,
mtouch_driver_funcs_t *funcs)
```

Arguments:

params

Configured parameters for your driver.

funcs

Callback functions that are implemented by the mtouch driver. The Input Events framework will use these callback functions to retrieve required information from the driver.

Library:

`libinputevents`

Description:

This function connects the driver with the specified parameters and callback functions to the Input Events frameowrk. You need to configure `params` and `funcs` before passing them as arguments to this function. This function must be called when you initialize your driver.

Returns:

A handle to the opaque data type that represents the touch device.

mtouch_driver_detach()

Detach driver from the Input Events framework.

Synopsis:

```
#include <input/mtouch_driver.h>
```

```
void mtouch_driver_detach(struct mtouch_device *device)
```

Arguments:

device

Handle to the touch device.

Library:

`libinputevents`

Description:

This function disconnects the specified driver from the Input Events framework. Any memory allocated by the framework to support this driver will be freed. This function must be called in addition to any device cleanup.

Returns:

Nothing.

mtouch_driver_process_packet()

Process the data packet from the specified driver.

Synopsis:

```
#include <input/mtouch_driver.h>

void mtouch_driver_process_packet(struct mtouch_device *device, void *packet,
 void *arg, unsigned int flags)
```

Arguments:

device

Handle to the touch device.

packet

Data packet that contains information on the touch-related event.

arg

User information that resulted from initializing your driver.

flags

Internal use only.

Library:

`libinputevents`

Description:

This function takes the data packet from the specfied driver and extracts information from it. The relevant information from the driver's data packet will be used to create an `mtouch_event_t` so that the Input Events framework can continue to process it and pass it to Screen. Drivers must call this function with a `packet` from which data for all digits currently in contact with the touchscreen can be retrieved. For example, if there are two digits in contact with the touchscreen, then information for the contact of both digits must be included in the single data packet. Drivers must not call this function multiple times when multiple digits are simultaneously touching. This situation would result in multiple touch and release events instead of an intended single move event.

Returns:

Nothing.

mtouch_test_bist()

Test the built-in-self-test (BIST)

Synopsis:

```
#include <input/mtouch_driver.h>

int mtouch_test_bist(void *dev, uint8_t *passfail, uint16_t *max, uint16_t
*min)
```

Arguments:

Library:

`libinputevents`

Description:

This function runs the BIST of the driver.

Returns:

A test result type. The test result will be one of type *Driver test result types* (p. 33).

mtouch_test_fini()

Clean up test driver interface.

Synopsis:

```
#include <input/mtouch_driver.h>

void mtouch_test_fini(void *dev)
```

Arguments:

dev

Handle to the device driver.

Library:

`libinputevents`

Description:

This function frees any memory allocated by the test driver interface used in running test functions.

Returns:

Nothing.

mtouch_test_init()

Initialize test driver interface.

Synopsis:

```
#include <input/mtouch_driver.h>

void* mtouch_test_init(void)
```

Arguments:

Library:

`libinputevents`

Description:

This function initializes the test driver with defaults and prepares the test driver interface for running individual test functions.

Returns:

A handle to the device driver. This handle is passed as an argument into each of the other test functions.

mtouch_test_read_firmware_version()

Test the ability to read the firmware version of the driver.

Synopsis:

```
#include <input/mtouch_driver.h>

int mtouch_test_read_firmware_version(void *dev, uint8_t **fwv)
```

Arguments:

dev

Handle to the device driver.

fwv

Firmware version read from the driver and is valid until the next call of any one of the test functions.

Library:

`libinputevents`

Description:

This function reads the firmware version from the driver. The first byte in `fwv` represents the number of additional bytes in the buffer.

Returns:

A test result type. The test result will be one of type *Driver test result types* (p. 33).

mtouch_test_read_product_id()

Test the ability to read the product ID of the driver.

Synopsis:

```
#include <input/mtouch_driver.h>

int mtouch_test_read_product_id(void *dev, uint8_t **product)
```

Arguments:

dev

Handle to the device driver.

product

product ID read from the driver. The product ID is updated by this function and is valid until the next call of any one of the test functions.

Library:

`libinputevents`

Description:

This function reads the product ID from the driver. The first byte in `product` represents the number of additional bytes in the buffer.

Returns:

A test result type. The test result will be one of type *Driver test result types* (p. 33).

mtouch_test_read_serial_id()

Test the ability to read the serial ID of the driver.

Synopsis:

```
#include <input/mtouch_driver.h>

int mtouch_test_read_serial_id(void *dev, uint8_t **serial)
```

Arguments:

dev

Handle to the device driver.

serial

Serial ID read from the driver. The serial ID is updated by this function and is valid until the next call of any one of the test functions.

Library:

`libinputevents`

Description:

This function reads the serial ID from the driver. The first byte in `serial` represents the number of additional bytes in the buffer.

Returns:

A test result type. The test result will be one of type *Driver test result types* (p. 33).

mtouch_test_read_supplier_id()

Test the ability to read the supplier ID.

Synopsis:

```
#include <input/mtouch_driver.h>

int mtouch_test_read_supplier_id(void *dev, uint8_t **supplier)
```

Arguments:

dev

Handle to the device driver.

supplier

Supplier ID read from the driver. The supplier ID is updated by this function and is valid until the next call of any one of the test functions.

Library:

`libinputevents`

Description:

This function reads the supplier ID from the driver. The first byte in `supplier` represents the number of additional bytes in the buffer.

Returns:

A test result type. The test result will be one of type *Driver test result types* (p. 33).

mtouch_log.h

Function and Macros that print mtouch log information.

Constants in *mtouch_log.h*

Constants that are available for logging input events.

Definitions in *mtouch_log.h*

Preprocessor macro definitions for the mtouch_log.h header file in the libinputevents library.

Definitions:

`#define mtouch_debug mtouch_log(_SLOG_DEBUG1, devname, format, ##args)`

Print mtouch log information as SLOG_DEBUG1 severity.

This macro implements an *mtouch_log()* function with severity of SLOG_DEBUG. If NDEBUG is defined, the information is sent as a message to the system logger (`slogger`). Otherwise, the log is simply directed to `stderr`. The *mtouch_log()* output format is: `devname[DEBUG]: formatted argument list`.

devname

Name of device driver.

format

String that specifies the format of the log. The formatting string determines what additional arguments you need to provide.

args

Variable-length argument list that correponds to that which is specified in `format`.

Nothing.

`#define mtouch_info mtouch_log(_SLOG_INFO, devname, format, ##args)`

Print mtouch log information as SLOG_INFO severity.

This macro implements an *mtouch_log()* function with severity of SLOG_INFO. If NDEBUG is defined, the information is sent as a message to the system logger (`slogger`). Otherwise, the log is simply directed to `stderr`. The *mtouch_log()* output format is: `devname[INFO]: formatted argument list`.

devname

Name of device driver.

format

String that specifies the format of the log. The formatting string determines what additional arguments you need to provide.

args

Variable-length argument list that correponds to that which is specified in `format`.

Nothing.

```
#define mtouch_warn mtouch_log(_SLOG_WARNING, devname, format, ##args)
```

Print mtouch log information as SLOG_WARNING severity.

This macro implements an *mtouch_log()* function with severity of SLOG_WARNING. If NDEBUG is defined, the information is sent as a message to the system logger (`slogger`). Otherwise, the log is simply directed to `stderr`. The *mtouch_log()* output format is: `devname[WARNING]: formatted argument list`.

devname

Name of device driver.

format

String that specifies the format of the log. The formatting string determines what additional arguments you need to provide.

args

Variable-length argument list that correponds to that which is specified in `format`.

Nothing.

```
#define mtouch_error mtouch_log(_SLOG_ERROR, devname, format, ##args)
```

Print mtouch log information as SLOG_ERROR severity.

This macro implements an *mtouch_log()* function with severity of SLOG_ERROR. If NDEBUG is defined, the information is sent as a message to the system logger (`slog`

ger). Otherwise, the log is simply directed to stderr. The *mtouch_log()* output format is: devname[ERROR]: formatted argument list.

devname

Name of device driver.

format

String that specifies the format of the log. The formatting string determines what additional arguments you need to provide.

args

Variable-length argument list that correponds to that which is specified in format.

Nothing.

#define mtouch_critical mtouch_log(_SLOG_CRITICAL, devname, format, ##args)

Print mtouch log information as SLOG_CRITICAL severity.

This macro implements an *mtouch_log()* function with severity of SLOG_ERROR. If NDEBUG is defined, the information is sent as a message to the system logger (slog ger). Otherwise, the log is simply directed to stderr. The *mtouch_log()* output format is: devname[CRITICAL]: formatted argument list.

devname

Name of device driver.

format

String that specifies the format of the log. The formatting string determines what additional arguments you need to provide.

args

Variable-length argument list that correponds to that which is specified in format.

Nothing.

Library:

libinputevents

Functions in *mtouch_log.h*

Constants and macros that are available for logging input events.

mtouch_log()

Print mtouch log information.

Synopsis:

```
#include <input/mtouch_log.h>

void mtouch_log(int severity, const char *devname, const char *format,...)
```

Arguments:

severity

Severity of the condition that triggered the log. For more information on severity levels, see *slogf()* in the *QNX C Library Reference*. All log severities are defined in `<sys/slog.h>`

devname

Name of device driver.

format

String that specifies the format of the log. The formatting string determines what additional arguments you need to provide.

...

Variable-length argument list that correponds to that which is specified in `format`.

Library:

`libinputevents`

Description:

This is a variadic function. If NDEBUG is defined, the information is sent as a message to the system logger (`slogger`). Otherwise, the log is simply directed to `stderr`. The *mtouch_log()* output format is: `devname[severity]: formatted argument list`.

Returns:

Nothing.

mtouch_params.h

Enumerations and Structures for mtouch parameters.

The `mtouch_params.h` header file provides type definitions related to mtouch parameters.

Constants in *mtouch_params.h*

Constants that are used to represent parmeters of a touch driver.

Definitions in *mtouch_params.h*

Preprocessor macro definitions for the mtouch_params.h header file in the libinputevents library.

Definitions:

```
#define MTOUCH_MAX_FILTERS 8
```

Maximum number of filters used by Screen.

This is somewhat arbitrary and is defined only so Screen knows how big to make its filters parameter array. There is no libinputevents limit to how many filters can be chained.

Library:

`libinputevents`

Data types in *mtouch_param.h*

Data structures, typedefs, and enumerations that are used to represent touch driver parameters.

mtouch_driver_params

Touch driver parameters.

Synopsis:

```
typedef struct mtouch_driver_params {
    _Uint32t capabilities ;
    _Uint32t flags ;
    _Uint8t max_touchpoints ;
    _Uint32t width ;
    _Uint32t height ;
    char vendor [MTOUCH_PARAMS_VENDOR_SZ];
    char product_id [MTOUCH_PARAMS_PRODUCT_ID_SZ];
    char sensor_id [MTOUCH_PARAMS_SENSOR_ID_SZ];
    _Uint32t sensor_sz_x ;
    _Uint32t sensor_sz_y ;
    _Uint32t max_refresh ;
}mtouch_driver_params_t;
```

Data:

_Uint32t capabilities

The capabilities supported by the driver.

Valid capabilities are:

- `MTOUCH_CAPABILITIES_CONTACT_ID`
- `MTOUCH_CAPABILITIES_COORDS`
- `MTOUCH_CAPABILITIES_CONTACT_COUNT`
- `MTOUCH_CAPABILITIES_WIDTH`
- `MTOUCH_CAPABILITIES_HEIGHT`
- `MTOUCH_CAPABILITIES_ORIENTATION`
- `MTOUCH_CAPABILITIES_PRESSURE`
- `MTOUCH_CAPABILITIES_RATE_SET`
- `MTOUCH_CAPABILITIES_SEQ_ID`
- `MTOUCH_CAPABILITIES_CONTACT_TYPE`
- `MTOUCH_CAPABILITIES_SELECT`

_Uint32t flags

Device flags.

Valid flags are:

- `MTOUCH_FLAGS_INCONSISTENT_DIGIT_ORDER`
- `MTOUCH_FLAGS_INCONSISTENT_CONTACT_IDS`

_Uint8t max_touchpoints

The maximum number of touchpoints supported by the driver.

_Uint32t width

The width, in touch units, of the touch area.

_Uint32t height

The height, in touch units, of the touch area.

char vendor[MTOUCH_PARAMS_VENDOR_SZ]

Internal use only.

char product_id[MTOUCH_PARAMS_PRODUCT_ID_SZ]

Internal use only.

char sensor_id[MTOUCH_PARAMS_SENSOR_ID_SZ]

Internal use only.

_Uint32t sensor_sz_x

Internal use only.

_Uint32t sensor_sz_y

Internal use only.

_Uint32t max_refresh

Internal use only.

Library:

`libinputevents`

Description:

mtouch_driver_params_t

Touch driver parameters.

Synopsis:

```
#include <input/mtouch_params.h>

typedef struct mtouch_driver_params  mtouch_driver_params_t;
```

Library:

`libinputevents`

Description:

This type is an alias for the touch driver parameters, mtouch_driver_params. Use this `typedef` when assigning parameters associated with your touch driver.

mtouch_filter_config

Touch filter configuration.

Synopsis:

```
typedef struct mtouch_filter_config {
    mtouch_filter_e type ;
    char * options ;
}mtouch_filter_config_t;
```

Data:

mtouch_filter_e type

Filter type.

char * options

Filter specific options.

The available filter options depend on the filter type.

Library:

`libinputevents`

Description:

mtouch_filter_config_t

Touch filter configuration.

Synopsis:

```
#include <input/mtouch_params.h>

typedef struct mtouch_filter_config  mtouch_filter_config_t;
```

Library:

`libinputevents`

Description:

This type is an alias for the filter configuration, mtouch_filter_config. Use this `typedef` when assigning the filter configuration associated with your driver.

mtouch_client_params

Touch client parameters.

Synopsis:

```
typedef struct mtouch_client_params {
    _Uint32t min_event_interval ;
    mtouch_scaling_params_t scaling ;
}mtouch_client_params_t;
```

Data:

_Uint32t min_event_interval

The sampling period (in microseconds) of the touch driver.

You will not get two touch samples within a time less than that configured here. The effectiveness of this configuration is driver-dependent.

mtouch_scaling_params_t scaling

The parameters for scaling touch coordinates.

Library:

`libinputevents`

Description:

mtouch_client_params_t

Touch client parameters.

Synopsis:

```
#include <input/mtouch_params.h>

typedef struct mtouch_client_params  mtouch_client_params_t;
```

Library:

`libinputevents`

Description:

This type is an alias for the client parameters, mtouch_client_params. Use this `typedef` when assigning client parameters.

parseopts.h

Helper functions to parse the options from the mtouch section of graphics.conf.

The `parseopts.h` header file defines the helper functions that are used to parse the options from the mtouch section of the configuration file, graphics.conf.

Functions in *parseopts.h*

Helper functions to parse the options from the `mtouch` section of `graphics.conf`.

The parseopts.h header file defines the helper functions that are used to parse the options from the `mtouch` section of the configuration file, `graphics.conf`.

input_parseopts()

Parse the argument `options`.

Synopsis:

```
#include <input/parseopts.h>

void input_parseopts(const char *options, set_option_t set_option, void *arg)
```

Arguments:

options

Option passed from `mtouch` section of `graphics.conf`.

set_option

Callback function that is called on each option in `options` to parse the information and set the user information.

arg

User information.

Library:

`libinputevents`

Description:

This function calls the `set_option` callback function on the device for each option in `options`. The format of `options` is: `option=value1,option2=value2`

Returns:

Nothing.

input_parse_bool()

Parse an `option` of the type boolean.

Synopsis:

```
#include <input/parseopts.h>

int input_parse_bool(const char *option, const char *value, unsigned *out)
```

Arguments:

option

Option of type boolean to be parsed.

value

Value of the option.

out

The boolean value of `option`.

Library:

`libinputevents`

Description:

Returns:

`0` if successful, otherwise non-zero if an error occurred.

input_parse_double()

Parse an `option` of the type double.

Synopsis:

```
#include <input/parseopts.h>

int input_parse_double(const char *option, const char *value, double *out)
```

Arguments:

option

Option of type string to be parsed.

value

Value of the option.

out

The double value of `option`.

Library:

`libinputevents`

Description:

Returns:

`0` if successful, otherwise non-zero if an error occurred.

input_parse_signed()

Parse an `option` of the type integer.

Synopsis:

```
#include <input/parseopts.h>

int input_parse_signed(const char *option, const char *value, int *out)
```

Arguments:

option

Option of type integer to be parsed.

value

Value of the option.

out

The integer value of `option`.

Library:

`libinputevents`

Description:

Returns:

`0` if successful, otherwise non-zero if an error occurred.

input_parse_string()

Parse an `option` of the type string.

Synopsis:

```
#include <input/parseopts.h>

int input_parse_string(const char *option, const char *value, char **out)
```

Arguments:

option

Option of type string to be parsed.

value

Value of the option.

out

The string value of `option`.

Library:

`libinputevents`

Description:

Returns:

`0` if successful, otherwise non-zero if an error occurred.

input_parse_unsigned()

Parse an `option` of the type unsigned integer.

Synopsis:

```
#include <input/parseopts.h>

int input_parse_unsigned(const char *option, const char *value, unsigned *out)
```

Arguments:

option

Option of type unsgined integer to be parsed.

value

Value of the option.

out

The unsigned integer value of `option`.

Library:

`libinputevents`

Description:

Returns:

`0` if successful, otherwise non-zero if an error occurred.

screen_helpers.h

Helper functions for Screen input events.

The `screen_helpers.h` header file provides functions for processing Screen Input events.

Functions in *screen_helpers.h*

Functions for working with screen input events.

screen_get_mtouch_event()

Retrieve mtouch event data from a screen event.

Synopsis:

```
#include <input/screen_helpers.h>

static int screen_get_mtouch_event(screen_event_t screen_event, mtouch_event_t
 *mtouch_event, int screen_abs)
```

Arguments:

screen_event

The Screen event to retrieve data from.

mtouch_event

The mtouch_event to populate.

screen_abs

The indicator to specify which coordinates to use; `1` indicates to use Screen coordinates. Otherwise, use source viewport.

Library:

`libinputevents`

Description:

The function `screen_get_mtouch_event()` populates the mtouch_event with data fetched from the Screen event.

Returns:

0 on success, -1 on failure.